Security through Distrusting

Joanna Rutkowska
Invisible Things Lab & Qubes OS Project

Black Hat EU, London, UK, December 7, 2017
Trust?

- Trusted(?)
- Secure
- Trustworthy

more desired
Trust consider harmful!
Security through Distrusting examples
Example #1: Pesky microphones
- Mics sniff our activities, including keystrokes, etc.
- Mics are difficult to neutralize
- Mics naturally “cross” security boundaries
- Mics sniff our activities, including keystrokes, etc.
- Mics are difficult to neutralize
- Mics naturally “cross” security boundaries
Example #2: Stateless laptop
Persistent laptop compromises...

- Persist
- Store secrets
- PII
- Persist
- Store secrets
- PII
- Persist
- Store secrets
- PII
Stateless Hardware
(persistent state eliminated)

- Firmware infections prevented
- No places to store stolen secrets
- Reliable way to verify firmware
- Reliable way to *choose* firmware
- Boot multiple environments
- Share laptops with others
Example #3: Multi-party signatures

Photo via Peter Todd (@petertoddbtc)
Mutli-sig does not need to involve multiple users!
Also: not just Bitcoin wallets...
Example #4: Binary (multi) signing
Why care about binary (multi-) signing?

- OS installation images
- Applications
- Updates
- Firmware
Prime target for backdooring!
Source code

Building

Binary (e.g. updates)

Distribution

Binary

Create
developer(s)

Sign
vendor

end user
Source code

Binary (e.g. updates)

Distribution (https, signed binaries)

Binary

Create

(... & sign!)

Sign

Verify

developer(s)

vendor

distributor

end user
Source code

Building
(eg. updates)

Binary

Distribution
(https, signed binaries)

Attack (back-/bug-door injection)

Create
(… & sign!)

developer(s)

Vendor

Verify

end user

get http://...

git pull & tag -v

... & sign!
Source code → Building (e.g. updates) → Binary (via git push) → Distribution (https, signed binaries) → Binary

- Git pull & tag -v
- wget http://...

Create (...& sign!)

Sign

Verify

developer(s)

vendor

end user
Source code

Building (git push)

Binary (e.g. updates)

Distribution (https, signed binaries)

Binary

Verify

End user

Vendor

Sign

Attacker (back-/bug-door injection)

Create (... & sign!)

Developer(s)

wget http/...

(git pull & tag -v)
Source code

Building (git push)

Binary (e.g. updates)

Distribution (https, signed binaries)

Source code

Binary

Verify

Create (& sign!)

Get http://...

(git pull & tag -y)

Sign

end user

vendor

developer(s)
**Source code**

- `wget http://...`
- `git pull & tag -v`

**Building**

- `git push`

**Vendor #1**

- `Binary (e.g. updates)`
- `Sign`

**Vendor #2**

- `Sign (https, signed binaries)`

**Vendor #3**

- `Sign`
- `One binary, multiple signatures!`

**Binary**

- `Verify`
- `end user`

**Distribution**

- `Create ...
  & sign!`
Multi-signed binaries

- Signed by people from different countries
- Different organizations (vendor & auditing)
- Signed by different *machines*
  - In the same organization
  - In different organization
https://reproducible-builds.org
Example 5: Preventing data leaks
Some software
( buggy/ backdoored or otherwise compromised)

Your data...!
Some software (buggy/backdoored or otherwise compromised)

Your data...!

VPN
Some software
( buggy/ backdoored or otherwise compromised)

Your data...!

VPN

Windows laptop (compromised or backdoored)
Some software (buggy/backdoored or otherwise compromised)

Your data...!

VPN

Different device (or VM?)

- Qubes TorVM (2011)
- Whonix (2012 – present)
- Whonix for Qubes (2014 – present)
- Tor-enabled routers (multiple projects/products)

Windows laptop (compromised or backdoored)
Cut off networking?

Some software

Not very useful...
Qubes OS templates

- Template VM
- Updates server
  - Download updates, etc
- App VM
  - User data
- No networking (no leaks)
Example #6: Compartimentalization
“Classic” compartmentalization...

Work VM

Personal VM
“Classic” compartmentalization...

...not very useful!
Work VM

Personal VM

...more useful...
Inter-compartments data transfers
Qubes PDF/Image converters:

- Very simple (& trusted) code!
- Very simple format (& easy to verify if indeed)
- Very complex format (risky to parse!)
- Very complex parsing (very risky!)
App sandboxing is just part of the story...
Networking stacks
USB & Bluetooth stacks
Graphics & UI
VPNs & firewalling
Corporate management
Root of trust (admin)

Work VM
Personal VM

PDF
JPG
MOV
Isolation is just part of the story!
ENCLAVE OPERATION

After AEX has completed, the logical processor is no longer in enclave mode and the exiting event is processed normally. Any new events that occur after the AEX has completed are treated as having occurred outside the enclave (e.g., a MMF in disassociating an interrupt handler).

3.2.3 Resuming Execution after AEX

After system software has serviced the event that caused the logical processor to enter an enclave, the logical processor can re-start execution using RESUME. RESUME restores registers and returns control to where execution was interrupted.

If the cause of the exit was an exception or a fault and was not resolved, the event will be triggered again. If the enclave is re-entered using RESUME, for example, if an enclave performs a divide by 0 operation, executing RESUME will cause the enclave to attempt to re-execute the failing instruction and result in another divide by 0 exception, in order to handle an exception that occurred inside the enclave, software can enter the enclave at a different location and make the exception handled within the enclave by executing the ENTR instruction. The exception handler within the enclave can attempt to resolve the failing condition or simply return and indicate to software that the enclave should be terminated (e.g., using BRK).

3.2.3.1 RESUME Interaction

RESUME restores registers depending on the mode of the enclave (32 or 64 bits):
- In 32-bit mode (IA32_EFER.LMA = 0 || CS.L = 0), the low 32-bits of the legacy registers (EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, SF, ZF, AF, PF, CF, and TF) are restored from the thread's GPR area of the current SSA frame. Neither the upper 32 bits of the legacy registers nor the 64-16 registers (RB...R15) are loaded.
- In 64 bit mode (IA32_EFER.LMA = 1 || CS.L = 1), all 64 bits of the general processor registers (RAX, RBX, RCA, RDS, RBDS, RDIP, RDI, RSI, RDI, R8...R15, RRF, and R12L) are loaded.

Extended features specified by the IA32_FEATURES MSR are restored from the thread's GPR area of the current SSA frame. The layout of the n89 area depends on the current values of IA32_EFER.LMA and CS.L:
- IA32_EFER.LMA = 0 || CS.L = 0
- 32-bit load in the same format that xsave/xfsave uses with these values
- IA32_EFER.LMA = 1 || CS.L = 1
- 64-bit load in the same format that xsave/xfsave uses with these values plus XEX.W = 1

3.3 CALLING ENCLAVE PROCEDURES

3.3.1 Calling Convention

In standard call conventions, subroutine parameters are generally pushed onto the stack. The called routine, being aware of its own stack layout, knows how to find parameter values based on compiler-time-computed offsets from the GP or BP register (depending on runtime conventions used by the compiler).

Because of the stack switch when calling an enclave, stack-based parameters cannot be found in this manner. Entering the enclave requires a modified parameter passing convention.

For example, the caller might push parameters onto the untrusted stack and then pass a pointer to those parameters to the enclave's API. The exact choice of calling conventions is up to the writer of the enclave routines; these routines can be hand-coded or compiler generated.

3.3.2 Register Preservation

As with most systems, it is the responsibility of the caller to preserve all registers except that used for returning a value. This is consistent with conventional usage and seeks to optimize the number of registers saved/restored opera-
Biggest challenge for Qubes OS is how to do desktop integration (seamless UX) without compromising isolation!
Example #7: Almighty admins?
Admins can steal all our data :(
Access to their data
Can’t modify policies
Can’t modify software/VM images
Can access and do *everything* she wants!

- Access to their data
- Can’t modify policies
- Can’t modify software/VM images
Hmm...
What we want instead:

- Access to their data
- Can’t modify policies
- Can’t modify software/VM images

- No access to user data
- Can modify policies
- Can install software/VM images

Admin → User

User → Admin
What we want instead:

- Access to their data
- Can’t modify policies
- Can’t modify software/VM images

Check our Qubes OS new Admin API for implementation details
Occasionally mishaps happen still...
Example #8: Plan B
Compromised machine -> Backup (files, disks) -> Clean machine
Compromised machine → Backup (files, disks) → Clean machine

Restoring compromised backup is risky!
Qubes (Paranoid) Backup Restore

Compromised Qubes:
- AppVM
- template (fedora)
- AppVM
- template (debian)
- AppVM
- template (whonix-*)
- dom0
- netvm
- usbvm

Fresh Qubes with compromised AppVMs restored:
- AppVM
- AppVM
- AppVM
- AppVM
- template (fedora)
- template (debian)
- template (whonix-*)
- dom0
- netvm
- usbvm

System reinstalltion
Paranoid mode backup restore
Security through Distrusting

- Division of Duty
  - Mics (#1)
  - Stateless laptop (#2)
  - Multi signatures (#3/4)
  - Tunneling (#5)

- Compartmentalization
  - Qubes (#6/7)
  - Tunneling (#5)
  - Qubes Backup Restore (#8)

- Plan B having
  - Qubes Backup Restore (#8)
Tradeoffs?
Usability!

Developer resources
(well thought architecture & APIs, difficulty adding new features)

Hardware resources & cost
(CPU, memory, disk)

- Compartmentalization
- Multisigs for binaries
- Stateless laptop (BOM costs)

- Mics
- Multisigs for wallets
- Compartmentalization (?)
Thanks!

***

https://qubes-os.org
https://invisiblethingslab.com
https://blog.invisiblethings.org
https://github.com/rootkovska
@QubesOS // Twitter for Qubes OS
@rootkovska // Personal Twitter
427F11FD0FAA4B080123F01CDDFA1A3E36879494 // Qubes Master Key
ED727C3O6E766BC85E621AA65FA6C3E4D9AFBB99 // Personal Master Key